Docket No. AUS920000757US1

METHOD AND APPARATUS FOR MANAGING TEXTURE MEMORY IN A DATA PROCESSING SYSTEM

BACKGROUND OF THE INVENTION

1. Technical Field:

The present invention relates generally to an improved data processing system, and in particular to a method and apparatus for managing graphic data. Still more particularly, the present invention relates to a method and apparatus managing texture memory in a data processing system.

2. Description of Related Art:

Data processing systems, such as personal computers and work stations, are commonly utilized to run computer-aided design (CAD) applications, computer-aided manufacturing (CAM) applications, and computer-aided software engineering (CASE) tools. Engineers, scientists, technicians, and others employ these applications daily. These applications involve complex calculations, such as finite element analysis, to model stress in structures. Other applications include chemical or molecular modeling applications. CAD/CAM/CASE applications are normally graphics intensive in terms of the information relayed to the user. Data processing system users may employ other graphics intensive applications, such as desktop publishing applications. Generally, users of these applications require and demand that the data processing systems be able to provide extremely fast graphics information.

The processing of a graphics data stream to provide a graphical display on a video display terminal requires an extremely fast graphics system to provide a display with a rapid response. In these types of graphics systems, primitives are received for processing and display. A primitive is a graphics element that is used as a building block for creating images, such as, for example, a point, a line, a triangle, a polygon, or a quadrilateral. A primitive is defined by a group of one or more vertices. A vertex defines a point, an end point of an edge, or a corner of a polygon where two edges meet. Data also is associated with a vertex in which the data includes information, such as positional coordinates, colors, normals, and texture coordinates. Commands are sent to the graphics system to define how the primitives and other data should be processed for display.

These commands are processed to generate two dimensional and three dimensional images. A texture is a digital representation of the surface of an object. In addition to two-dimensional qualities, such as color and brightness, the texture is also encoded with three-dimensional properties, such as how transparent and reflective the object is. Once the texture has been defined, the texture can be wrapped around any 3-dimensional object. This process is called texture mapping.

Well-defined textures are very important for rendering realistic 3-D images. However, these textures also require a lot of memory, so they are not used as often as they might be. This requirement is one of the

rationales for the development of the new graphics interface, advanced graphic port (AGP), which allows texture to be stored in main memory, which is more expansive than video memory. AGP also speeds up the transfer of large textures between memory, the CPU and the video adapter.

Currently, systems allow applications to use both video memory on the graphics adapter as well as main memory to store and use textures. Additionally, applications and application program interfaces (APIs) may use this texture memory, which is made up of video memory and AGP memory. OpenGL and Direct3D are examples of code operating in different levels of the operating system. OpenGL runs in application space, also referred to as "client space", and Direct3D operates in kernel space. OpenGL is a graphics programming language by Silicon Graphics Incorporated. Direct3D is an application program interface (API) for manipulating and displaying 3-dimensional objects, developed by Microsoft Corporation. Each of these software systems pre-allocates a large amount of texture memory regardless of how much memory is required.

Such a mechanism is inefficient because the amount of memory required is unknown and a condition may exist in which one application has insufficient memory, while another application wastes extra, unneeded memory.

Therefore, it would be advantageous to have an improved method and apparatus for managing texture memory.

SUMMARY OF THE INVENTION

The present invention provides a method, apparatus, and computer implemented instructions for managing a set of memory resources used to store texture objects in a data processing system. A texture manager allocates memory to a current texture object in a set of memory resources. A stored texture object, handled by the texture manager, is selectively removed in response to an inability to allocate sufficient memory to the current texture object. The allocating and selectively removing steps are repeated until the current texture object is allocated sufficient memory. The repeating step is halted in response to an absence of any stored texture objects, handled by a texture manager, being present in the first memory resource. Stored texture objects, handled by another texture manager, are selectively removed in response to an inability to allocate sufficient memory to the current texture object. Memory is allocated in the set of memory resources to the current texture object in response to selectively removing stored texture objects.

BRIEF DESCRIPTION OF THE DRAWINGS

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself, however, as well as a preferred mode of use, further objectives and advantages thereof, will best be understood by reference to the following detailed description of an illustrative embodiment when read in conjunction with the accompanying drawings, wherein:

Figure 1 is a pictorial representation of a data processing system in which the present invention may be implemented in accordance with a preferred embodiment of the present invention;

Figure 2 is a block diagram of a data processing system in which the present invention may be implemented;

Figure 3 is a diagram of a texture management system in accordance with a preferred embodiment of the present invention;

Figure 4 is a flowchart of a process used for managing texture memory in accordance with a preferred embodiment of the present invention; and

Figure 5 is a flowchart of a process used for reloading textures in accordance with a preferred embodiment of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

With reference now to the figures and in particular with reference to **Figure 1**, a pictorial representation of a data processing system in which the present invention may be implemented is depicted in accordance with a preferred embodiment of the present invention. A computer **100** is depicted which includes a system unit **110**, a video display terminal **102**, a keyboard **104**, storage devices **108**, which may include floppy drives and other types of permanent and removable storage media, and mouse **106**. Additional input devices may be included with personal computer **100**, such as, for example, a joystick, touchpad, touch screen, trackball, microphone, and the like. Computer **100** can be implemented using any suitable computer, such as an IBM RS/6000 computer or IntelliStation computer, which are products of International Business Machines Corporation, located in Armonk, New York. Although the depicted representation shows a computer, other embodiments of the present invention may be implemented in other types of data processing systems, such as a network computer. Computer **100** also preferably includes a graphical user interface that may be implemented by means of systems software residing in computer readable media in operation within computer **100**.

With reference now to **Figure 2**, a block diagram of a data processing system is shown in which the present invention may be implemented. Data processing system **200**

is an example of a computer, such as computer **100** in **Figure 1**, in which code or instructions implementing the processes of the present invention may be located. Data processing system **200** employs a peripheral component interconnect (PCI) local bus architecture. Although the depicted example employs a PCI bus, other bus architectures such as Accelerated Graphics Port (AGP) and Industry Standard Architecture (ISA) may be used. Processor **202** and main memory **204** are connected to PCI local bus **206** through PCI bridge **208**. Portions of main memory **204** may be allocated as texture memory, such as AGP memory, which may be used by applications and APIs for storing textures. PCI bridge **208** also may include an integrated memory controller and cache memory for processor **202**. Additional connections to PCI local bus **206** may be made through direct component interconnection or through add-in boards.

In the depicted example, local area network (LAN) adapter **210**, small computer system interface SCSI host bus adapter **212**, and expansion bus interface **214** are connected to PCI local bus **206** by direct component connection. In contrast, audio adapter **216**, graphics adapter **218**, and audio/video adapter **219** are connected to PCI local bus **206** by add-in boards inserted into expansion slots. Expansion bus interface **214** provides a connection for a keyboard and mouse adapter **220**, modem **222**, and additional memory **224**. SCSI host bus adapter **212** provides a connection for hard disk drive **226**, tape drive **228**, and CD-ROM drive **230**. For example, graphics adapter **218** contains video memory, which also may be used to store textures.

An operating system runs on processor **202** and is used to coordinate and provide control of various components within data processing system **200** in **Figure 2**. The operating system may be a commercially available operating system such as Windows 2000, which is available from Microsoft Corporation. An object oriented programming system such as Java may run in conjunction with the operating system and provides calls to the operating system from Java programs or applications executing on data processing system **200**. "Java" is a trademark of Sun Microsystems, Inc. Instructions for the operating system, the object-oriented programming system, and applications or programs are located on storage devices, such as hard disk drive **226**, and may be loaded into main memory **204** for execution by processor **202.**

Those of ordinary skill in the art will appreciate that the hardware in **Figure 2** may vary depending on the implementation. Other internal hardware or peripheral devices, such as flash ROM (or equivalent nonvolatile memory) or optical disk drives and the like, may be used in addition to or in place of the hardware depicted in **Figure 2**. Also, the processes of the present invention may be applied to a multiprocessor data processing system.

For example, data processing system **200**, if optionally configured as a network computer, may not include SCSI host bus adapter **212**, hard disk drive **226**, tape drive **228**, and CD-ROM **230**, as noted by dotted line **232** in **Figure 2** denoting optional inclusion. In that case, the computer, to be properly called a client computer, must include some type of network communication

interface, such as LAN adapter **210**, modem **222**, or the like. As another example, data processing system **200** may be a stand-alone system configured to be bootable without relying on some type of network communication interface, whether or not data processing system **200** comprises some type of network communication interface. As a further example, data processing system **200** may be a personal digital assistant (PDA), which is configured with ROM and/or flash ROM to provide non-volatile memory for storing operating system files and/or user-generated data.

The depicted example in **Figure 2** and above-described examples are not meant to imply architectural limitations. For example, data processing system **200** also may be a notebook computer or hand held computer in addition to taking the form of a PDA. Data processing system **200** also may be a kiosk or a Web appliance.

The processes of the present invention are performed by processor **202** using computer implemented instructions, which may be located in a memory such as, for example, main memory **204**, memory **224**, or in one or more peripheral devices **226-230**.

The present invention provides a method, apparatus, and computer implemented instructions for managing textures and texture memory in a data processing system, such as data processing system **200**. The mechanism of the present invention provides a seamless integration of texture memory for use by applications and APIs operating at different levels within an operating system.

Turning next to **Figure 3**, a diagram of a texture management system is depicted in accordance with a

preferred embodiment of the present invention. In this example, texture manager **300**, texture manager **302**, and miniport **304** are components used to manage video memory **306** and AGP memory **308**. Miniport **304** serves as a center point to allocate and free texture memory within video memory **306** and AGP memory **308**. In other words, miniport **304** is a memory resource manager.

In the depicted examples, texture manager **300** is associated with OpenGL application **310** and located in client space **312**. Texture manager **302** is associated with Direct3D API **314** and is located within kernel space **316**. Video memory **306** and APG memory **308** are located in hardware **318**. Specifically, video memory 306 may be, for example, a frame buffer and a graphic adapter, such as graphic adapter **218** in **Figure 2**. APG memory **308** may be located within main memory **204** in **Figure 2**.

Although this example illustrates the use of an OpenGL application and a Direct3D API, the mechanism of the present invention may be applied to other applications and APIs operating in client space **312** and kernel space **316**.

Each texture manager, texture manager **300** and texture manger **302**, allocates and frees texture memory through commands to miniport **304**. This is performed by each texture manager as if each one owned all of the texture manager resources. The actual allocation and free of texture memory is performed by miniport **304**. Each texture manager keeps track of its allocations for all textures. If miniport **304** fails to allocate a texture memory as requested, the texture manager boots

the least active texture to make room for the new allocations. After freeing, miniport **304** is again called to allocate texture memory. The process continues until the allocation succeeds or no more textures are present to be freed. If no more texture present, texture manager **302** returns a failure. On the other hand, texture manager **300** will free up memory by removing textures handled by texture manager **302**. In these examples, texture manager **300** has a higher priority over texture manager **302** and is allowed to remove textures handled by texture manager **302**.

Turning next to **Figure 4**, a flowchart of a process used for managing texture memory is depicted in accordance with a preferred embodiment of the present invention. The process illustrated in **Figure 4** may be implemented in a texture manager, such as texture manager **300** or **302** in **Figure 3**. This process is applied to both video memory and memory in the data processing system for storing textures.

The process begins by receiving a request to store a texture in texture memory (step **400**). A texture is reloaded as needed (step **402**). Texture memory is then allocated to the current texture in the request (step **404**). In this example, the allocation occurs through a call made by the texture manager to a miniport or other memory allocation mechanism. A determination is made as to whether the allocation was successful (step **406**). In this example, allocations are done a per mipmap basis. A texture object consists of one or more mipmaps starting from level 0 and the allocation always begins with mipmap

0, then 1, and so on. A mipmap is a reduced resolution version of a texture map, used to texture a geometric primitive whose screen resolution differs from the resolution of the source texture map.

The allocation is successful if sufficient memory is present to store the current texture. If the allocation is successful, the process terminates.

If the allocation is unsuccessful, a determination is made as to whether all textures have been removed from the texture memory (step **408**). This determination in step **406** is made with respect to texture objects handled or tracked by the texture manager.

If one or more textures are still present in the texture memory, then a less active texture is removed (step **410**). A less active texture may be identified by comparing the use of a particular texture in rendering an object to a threshold. Each time a texture is activated, it is assigned a unique global value, which gets decreased by one after each use. This value will be used to determine how active a texture relative to another texture by comparing the value of this texture to the value of the other texture. A texture with largest number is the least active one.

The removed texture is marked (step **412**). If a texture is removed from APG memory, this texture is marked as a APG texture such that when it is reactivated or loaded again, this texture will be loaded into APG memory. If the texture is removed from video memory, it is marked as a video texture so that this texture will be reloaded into video memory the next time it is requested for use. Memory is allocated to the current texture

object (step **414**) with the process then returning to step **406** as described above.

With reference again to step **408**, if all of the texture objects have been removed from texture memory, then allocation of memory to the current texture object is freed (step **416**). This step occurs because if the allocation fails after all textures are freed, a fragmented memory situation may be present. Then, memory is reallocated to the current texture (step **418**). In this case, steps **416** and **418** free memory allocated to the texture and reperforms the allocation starting from mipmap level zero. A texture object consists of one or more mipmaps starting from level 0 and the allocation always begins with mipmap 0, then 1, and so on.

Because texture objects are being allocated and freed, this could cause fragmentation in the memory and a mipmap level 0 could be allocated anywhere in the memory while next mipmaps (of the same object) could be allocated far from it. This allocation may lead to failure of subsequent mipmap allocation even though the total available space is larger than the sum of all mipmaps of the texture object.

In this case, the current object being allocated will be freed and re-allocated from mipmap level 0. This process will allocate more mipmaps sequentially and thus, the entire object may be allocated successfully without further work.

Thereafter, a determination is made as to whether the allocation is successful (step **420**). If the allocation is unsuccessful, textures allocated to lower priority processes are removed (step **422**) with the

process then returning to step **416** as described above. A single texture may be removed or all of the textures may be removed in step **422** depending on the particular implementation. These processes may be applications or APIs. For example, an OpenGl application has a higher priority than a Direct3D API. Thus, a texture manager associated with the OpenGL application would be allowed to remove textures allocated for the Direct3D API by the texture manager associated with this API.

If the allocation is successful, the process terminates. If this process is implemented in a texture manager associated with a lower priority process, then step **422** is replaced with a step in which an error or message indicating an allocation failure is returned.

Turning now to **Figure 5**, a flowchart of a process used for reloading textures is depicted in accordance with a preferred embodiment of the present invention. The process illustrated in **Figure 5** is a more detailed illustration of step **402** in **Figure 4**. The process begins by receiving a request to load a texture (step **500**). A determination is made as to whether a texture is marked with a memory type (step **502**). If the texture is marked with a memory type, then the texture is loaded into the type of memory indicated (step **504**) with the process terminating thereafter. For example, if the texture is marked as being a video texture, then the texture is placed into video memory. If the texture is marked as an AGP texture, the texture is placed into AGP memory.

With reference again to step **502**, if the texture is not marked with a memory type, the texture is then processed and placed into a memory (step **506**) with the

process terminating thereafter. A number of reasons exist for assigning a texture to a memory type. One is as a request by an application. This reason is a Direct 3D feature. Otherwise the assignment is determined by texture manager based on the size of the texture. Large sized textures are placed in AGP memory so that they will not be required swapped as often. When a less active texture is swapped out of video memory, and if it is not requested to be in video memory, this texture assigned to AGP memory. The next time the texture is used, this texture would be re-loaded into AGP memory.

Thus the present invention provides an improved method and apparatus for managing texture memory. This mechanism allows for reallocation of memory between different processes with may operated at different levels in an operating system. The mechanism involves freeing or removing textures handled by a texture manager to allocate memory for a current or new texture that is to be used in rendering an image. If all of the textures handled by a first texture manager are freed, the first texture manager may instruct a second texture manager to remove textures handled by it if the first texture manager has priority over the second texture manager. If a priority does not exist, a message or error may be returned to indicate a failure in allocating memory for a texture.

Although the depicted examples show an OpenGL application having priority over a Direct3D API, the priorities may be reversed to give the Direct3D API priority.

It is important to note that while the present

invention has been described in the context of a fully functioning data processing system, those of ordinary skill in the art will appreciate that the processes of the present invention are capable of being distributed in the form of a computer readable medium of instructions and a variety of forms and that the present invention applies equally regardless of the particular type of signal bearing media actually used to carry out the distribution. Examples of computer readable media include recordable-type media, such as a floppy disk, a hard disk drive, a RAM, CD-ROMs, DVD-ROMs, and transmission-type media, such as digital and analog communications links, wired or wireless communications links using transmission forms, such as, for example, radio frequency and light wave transmissions. The computer readable media may take the form of coded formats that are decoded for actual use in a particular data processing system.

The description of the present invention has been presented for purposes of illustration and description, and is not intended to be exhaustive or limited to the invention in the form disclosed. Many modifications and variations will be apparent to those of ordinary skill in the art. The embodiment was chosen and described in order to best explain the principles of the invention, the practical application, and to enable others of ordinary skill in the art to understand the invention for various embodiments with various modifications as are suited to the particular use contemplated.